Raphael Giraud, Kersual Arnaud
GROUP : B2
19/01/2023

Compte_rendu
S1.01

Table des matières

Introduction

The project named “Automatic Classification” have a goal.

It is to create a program as reliable and as fast as possible to automatically classify a number of dispatch(Which will be optimized to be more precise).

The program will assign each dispatches to one of the following five categories :

- ENVIRONNEMENT-SCIENCES

- CULTURE

- ECONOMIE

- POLITIQUE

- SPORTS

To perform this classification, the program will need to rely on key words that belongs to a single category.

In addition each word have a weight : 1,2 or 3 (3 is the maximum) meaning the word is strongly marking it’s belonging to a category.

The project is split in two parts: The first one is to manually create lexicons for each categories plus functions to categorize dispatches according to the lexicons.

For each dispatch, a score is calculated for each categories (5 then).

The second part of the project is the same but automatically, we generate a lexicon or several.

The goal is to perform a machine learning methode and optimize the code to be faster and more efficient.

Finally, the last part of the project is to calculate the percentage of success for each category and write to a file the information that will be created by the program.

What we have done

We have finished the first part and the second part.

We couldn't add other learning data with the rss feed unfortunately.

We successfully created a ranking in percentage, of the success of each categories then written in a file.

And an average of the total success of the word lexicons.

For the second part we automatically generated lexicons for each categories. And optimized it, to take the most representative words of one category.

And then created a file that contains the result of the program

For the first part we've done the following functions :

- initLexique

- entierPourChaine

- score

- chaineMax

- indicePourChaine

- moyenne

- classementDepeches

For the second part we've done the following functions :

- InitDico

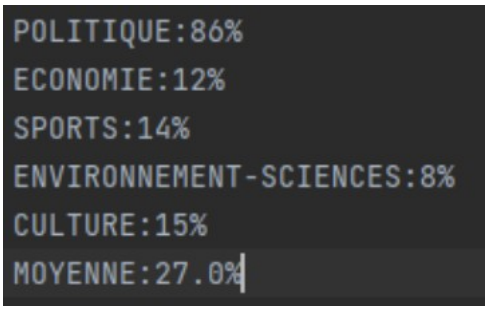
- CalculScores

- PoidsPourScore

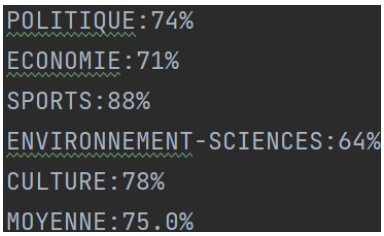
- GenerationLexique

Our result

For the first part we have an average of […] % of success in all categories for the lexicons written manually, those result are low because the lexicons for them are not efficient enough.



For the second part we have an average of 75 % of success in all categories for lexicons that were auto-generated according to the frequency of apperance of the words in the dispatches of a same category.



These result are more precise than the manual program.

In fact the lexicon created by this program is less concise and help a lot in those result.

The amount of information stored in it give a larger field for it to compare to other dispatches.

Also the fact that the lexicon list is sorted and that the search is dichotomique is a big factor in gaining time.

So the average of success will be better.

Analysis of complexity

We will analyze the time taken by the algorithm in the worst case :

Worst case : each word is the last position for “entierPourChaine”

```java
public int score(Depeche d) {
    int somme = 0;
    for (String mot : d.getMots()) {
        somme += UtilitairePaireChaineEntier.entierPourChaine(lexique, mot, gauche: 0, droite: lexique.size()-1);
    }
    return somme;
}
```

The algorithm is logarithmic because of it’s dickotous research running with Utilitaire so, the complexity is : (n)=O(n)(log(n))

```java
private static void fusion(ArrayList<PaireChaineEntier> gauche, ArrayList<PaireChaineEntier> droite, ArrayList<PaireChaineEntier> atrie) {
    int i = 0;
    int j = 0;
    int k = 0;
    while (i < gauche.size() && j < droite.size()) {
        if (gauche.get(i).getChaine().compareTo(droite.get(j).getChaine()) < 0) {
            atrie.set(k++, gauche.get(i++));
        } else {
            atrie.set(k++, droite.get(j++));
        }
    }
    while (i < gauche.size()) {
        atrie.set(k++, gauche.get(i++));
    }
    while (j < droite.size()) {
        atrie.set(k++, droite.get(j++));
    }
}
```

The algorithm called with the first one is also a logarithmic one with a complexity of : n= n(log(n))

```java
public static void calculScores(ArrayList<Depeche> depeches, String categorie, ArrayList<PaireChaineEntier> dictionnaire) {
    for (PaireChaineEntier paire : dictionnaire) {
        for (Depeche depeche : depeches) {
            if (depeche.getCategorie().equals(categorie) && depeche.getMots().contains(paire.getChaine())) {
                paire.setEntier(paire.getEntier() + 1);
            } else if (!depeche.getCategorie().equals(categorie) && depeche.getMots().contains(paire.getChaine())) {
                paire.setEntier(paire.getEntier() - 2);
            }
        }
    }
}
```

The second algorithm is linear because of it’s multiple loops that are running inside the function so, the complexity is : (n)=O(n^^2)

Conclusion

We can upgrade even further our program.

For the “poidspourScore” method and “CalculScore” of the Classification class, the coefficients can be upgraded.

Indeed to find the ideal weight to assign to a given score, we will need to test a large amount of times our project with different coefficients and apply the one that fits the most.

This method should also be applied to determine the best coefficients for the function “CalculScore”.

For now these coefficients are +1 if a word is present and + 1 each times it is in the given category, and -2 if it is present in another category than the given one.